Object-Oriented Programming I (CSE 121)
Week # 06
(YouTube Lectures: 10, 11 & 12)
Arrays in C++

Objective(s)

In this chapter, you will learn:

- Array declaration, initialization and accessing array elements
- Examples of 1D, 2D and 3D arrays
- Passing arrays to a function

- An array stores multiple values in a single variable, instead of declaring separate variables for each value.
- An array is a collection of elements of the same type placed in contiguous memory locations that can be individually referenced by using an index to a unique identifier.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

There are 2 types of array in C++:

1. One-dimensional array (1D array)
2. Multi-dimensional array (2D array, 3D array, …)

One-dimensional Array Declaration (1D Array)

`dataType` arrayName[arraySize];

For example, `int` x[6];

Here,

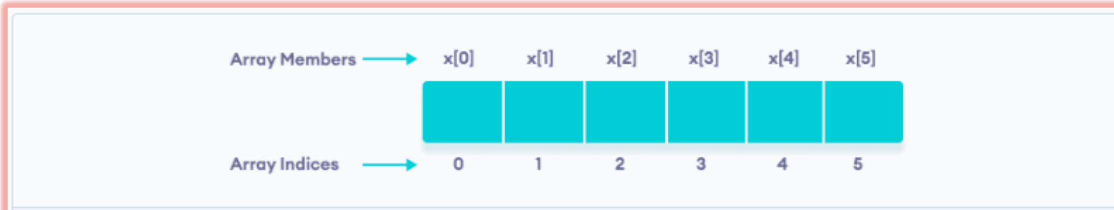
- `int` - type of element to be stored
- `x` - name of the array
- `6` - size of the array

Access Elements in C++ Array

In C++, each element in an array is associated with a number. The number is known as an array index. We can access elements of an array by using those indices.

```
arrayName[index]; // syntax to access array elements
```

Consider the array `x` we have seen above.



Elements of an array in C++

Few Things to Remember:

- The array indices start with `0`. Meaning `x[0]` is the first element stored at index `0`.
- If the size of an array is `n`, the last element is stored at index `(n-1)`. In this example, `x[5]` is the last element.

Bulbul Ahamed, Associate Professor of CSE, SU, bulbul2767@gmail.com

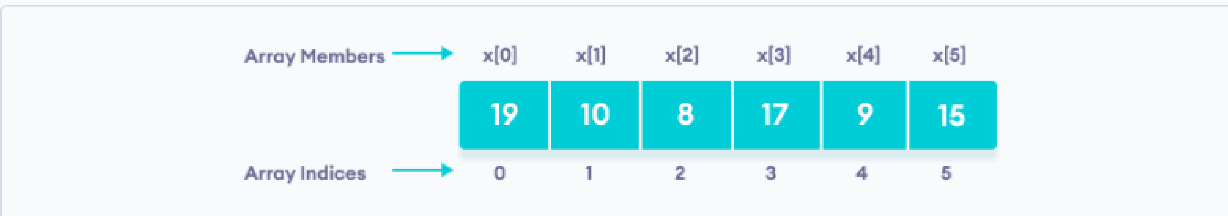
- Elements of an array have consecutive addresses. For example, suppose the starting address of x[0] is 2120d. Then, the address of the next element x[1] will be 2124d, the address of x[2] will be 2128d and so on.

Here, the size of each element is increased by 4. This is because the size of int is 4 bytes.

C++ Array Initialization

In C++, it's possible to initialize an array during declaration. For example,

```
int x[6] = {19, 10, 8, 17, 9, 15}; // declare and initialize an array
```



C++ Array elements and their data

Another method to initialize array during declaration:

```
int x[] = {19, 10, 8, 17, 9, 15}; // declare and initialize an array
```

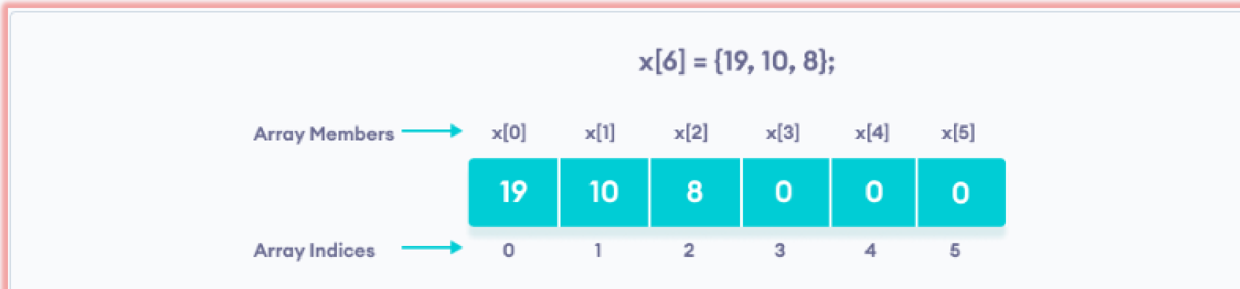
Here, we have not mentioned the size of the array. In such cases, the compiler automatically computes the size.

C++ Array with Empty Members

In C++, if an array has a size n, we can store up to n number of elements in the array. However, what will happen if we store less than n number of elements. For example,

```
int x[6] = {19, 10, 8}; // store only 3 elements in the array
```

Here, the array x has a size of 6. However, we have initialized it with only 3 elements.
In such cases, the compiler assigns random values to the remaining places. Oftentimes, this random value is simply 0.



Empty array members are automatically assigned the value 0

Program: Displaying array elements using for loop

one.cpp

```cpp
#include <iostream>
using namespace std;

int main() {
    int x[6] = {19, 10, 8, 17, 9, 15};

    cout << "The numbers are: ";
    for (int i = 0; i < 6; ++i) {
        cout << x[i] << "  ";
    }
    return 0;
}
```

C:\Users\Bulbul\Desktop\Array\one.exe

```
The numbers are: 19  10  8  17  9  15
Process returned 0 (0x0)   execution time : 0.091 s
Press any key to continue.
```

Program: Displaying array elements using range based for loop

two.cpp

```cpp
#include <iostream>
using namespace std;

int main() {
    int x[6] = {19, 10, 8, 17, 9, 15};

    cout << "The numbers are: ";
    for (int n : x) {
        cout << n << "  ";
    }
    return 0;
}
```

C:\Users\Bulbul\Desktop\Array\one.exe

```
The numbers are: 19  10  8  17  9  15
Process returned 0 (0x0)   execution time : 0.091 s
Press any key to continue.
```

Program: Take inputs from user and store them in an array

three.cpp

```cpp
#include <iostream>
using namespace std;

int main() {
    int x[6];
    cout << "Enter 6 numbers: ";
    for (int i = 0; i < 6; i++){
        cin >> x[i];
    }
    cout << "The numbers are: ";
    for (int i = 0; i < 6; i++){
        cout << x[i] << " ";
    }
    return 0;
}
```

C:\Users\Bulbul\Desktop\Array\three.exe

```
Enter 6 numbers: 19 10 8 17 9 15
The numbers are: 19 10 8 17 9 15
Process returned 0 (0x0)   execution time : 22.330 s
Press any key to continue.
```

Program: Find Sum and Average of array elements using range based for loop

four.cpp

```cpp
#include <iostream>
using namespace std;

int main() {
    double x[] = {19, 10, 8, 17, 9, 15};
    double sum = 0, count = 0, average;

    cout << "The numbers are: ";
    for (double const &n: x){
        cout << n << " ";
        sum += n;
        count++;
    }
    cout << "\nTheir sum = " << sum << endl;
    average = sum / count;
    cout << "Their average = " << average << endl;
    return 0;
}
```

C:\Users\Bulbul\Desktop\Array\four.exe

```
The numbers are: 19 10 8 17 9 15
Their sum = 78
Their average = 13
```

Passing One-dimensional Array to a Function

In C++, we can pass arrays as an argument to a function. And, also we can return arrays from a function.

Syntax for Passing Arrays as Function Parameters

The syntax for passing an array to a function is:

```
returnType functionName(dataType arrayName[arraySize]) {
   // code
}
```

Let's see an example,

```
int total(int marks[5]) {
   // code
}
```

Here, we have passed an int type array named marks to the function total(). The size of the array is 5.

Example: Passing one-dimensional array to a function

eight.cpp

```cpp
// C++ Program to display marks of 5 students
#include <iostream>
using namespace std;

void display(int m[5]) {
    cout << "Displaying marks: " << endl;
    // display array elements
    for (int i = 0; i < 5; ++i) {
        cout << "Student " << i + 1 << ": " << m[i] << endl;
    }
}
int main() {
    // declare and initialize an array
    int marks[5] = {88, 76, 90, 61, 69};
    // pass array as argument
    display(marks);
    return 0;
}
```

C:\Users\Bulbul\Desk

```
Displaying marks:
Student 1: 88
Student 2: 76
Student 3: 90
Student 4: 61
Student 5: 69
```

Here,

1. When we call a function by passing an array as the argument, only the name of the array is used.

   ```
   display(marks);
   ```

   Here, the argument marks represent the memory address of the first element of array marks[5].
2. However, notice the parameter of the display() function.

   ```
   void display(int m[5])
   ```

   Here, we use the full declaration of the array in the function parameter, including the square braces [].
3. The function parameter int m[5] converts to int* m;. This points to the same address pointed by the array marks. This means that when we manipulate m[5] in the function body, we are actually manipulating the original array marks.

C++ Multidimensional Arrays

In C++, we can create an array of an array, known as a multidimensional array. For example:

```
int x[3][4];
```

Here, x is a two-dimensional array. It can hold a maximum of 12 elements.

We can think of this array as a table with 3 rows and each row has 4 columns as shown below.

	Col 1	Col 2	Col 3	Col 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Elements in two dimensional array in C++ Programming

Three-dimensional arrays also work in a similar way. For example:

```
float x[2][4][3];
```

This array x can hold a maximum of 24 elements.

We can find out the total number of elements in the array simply by multiplying its dimensions:

```
2 x 4 x 3 = 24
```

Multidimensional Array Initialization

Like a normal array, we can initialize a multidimensional array in more than one way.

1. Initialization of two-dimensional array (2D Array)

```
int test[2][3] = {2, 4, 5, 9, 0, 19};
```

The above method is not preferred. A better way to initialize this array with the same array elements is given below:

```
int test[2][3] = { {2, 4, 5}, {9, 0, 19} };
```

This array has 2 rows and 3 columns, which is why we have two rows of elements with 3 elements each.

	Col 1	Col 2	Col 3
Row 1	2	4	5
Row 2	9	0	19

Initializing a two-dimensional array in C++

2. Initialization of three-dimensional array (3D Array)

```
int test[2][3][4] = {3, 4, 2, 3, 0, -3, 9, 11, 23, 12, 23,
                     2, 13, 4, 56, 3, 5, 9, 3, 5, 5, 1, 4, 9};
```

This is not a good way of initializing a three-dimensional array. A better way to initialize this array is:

```
int test[2][3][4] = {
                        { {3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2} },
                        { {13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9} }
                    };
```

Example: Displaying two-dimensional array elements

five.cpp

```
#include <iostream>
using namespace std;

int main() {
    int test[3][2] = {{2, -5},{4, 0},{9, 1}};
    // access rows of the array
    for (int i = 0; i < 3; ++i) {
        // access columns of the array
        for (int j = 0; j < 2; ++j) {
            cout << "test[" << i << "][" << j << "] = " << test[i][j] << endl;
        }
    }
    return 0;
}
```

C:\Users\Bulbul\De

```
test[0][0] = 2
test[0][1] = -5
test[1][0] = 4
test[1][1] = 0
test[2][0] = 9
test[2][1] = 1
```

The above example, we have initialized a two-dimensional `int` array named `test` that has 3 "rows" and 2 "columns".

Here, we have used the nested `for` loop to display the array elements.

- the outer loop from `i = 0` to `i = 2` access the rows of the array
- the inner loop from `j = 0` to `j = 1` access the columns of the array

Finally, we print the array elements in each iteration.

Example: Taking inputs from user for two-dimensional array

six.cpp

```cpp
#include <iostream>
using namespace std;

int main() {
    int numbers[2][3];
    cout << "Enter 6 numbers: " << endl;
    // Storing user input in the array
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 3; ++j) {
            cin >> numbers[i][j];
        }
    }
    cout << "The numbers are: " << endl;
    //  Printing array elements
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 3; ++j) {
            cout << "numbers[" << i << "][" << j << "]: " << numbers[i][j] << endl;
        }
    }
    return 0;
}
```

C:\Users\Bulbul\Desktop

```
Enter 6 numbers:
1 2 3 4 5 6
The numbers are:
numbers[0][0]: 1
numbers[0][1]: 2
numbers[0][2]: 3
numbers[1][0]: 4
numbers[1][1]: 5
numbers[1][2]: 6
```

Here, we have used a nested `for` loop to take the input of the 2d array. Once all the input has been taken, we have used another nested `for` loop to print the array members.

Example: Displaying three-dimensional array elements

seven.cpp

```cpp
#include <iostream>
using namespace std;

int main() {
    int test[2][3][2] = { { {1, 2}, {3, 4}, {5, 6} },
                          { {7, 8}, {9, 10}, {11, 12} }
                        };
    // Displaying the values with proper index.
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 3; ++j) {
            for (int k = 0; k < 2; ++k) {
                cout << "test[" << i << "][" << j << "][" << k << "] = " << test[i][j][k] << endl;
            }
        }
    }
    return 0;
}
```

C:\Users\Bulbul\Desktop\

```
test[0][0][0] = 1
test[0][0][1] = 2
test[0][1][0] = 3
test[0][1][1] = 4
test[0][2][0] = 5
test[0][2][1] = 6
test[1][0][0] = 7
test[1][0][1] = 8
test[1][1][0] = 9
test[1][1][1] = 10
test[1][2][0] = 11
test[1][2][1] = 12
```

- A 3D array is that array whose elements are two-dimensional arrays.
- The basic concept of printing elements of a 3D array is similar to that of a 2D array.
- However, since we are manipulating 3 dimensions, we use a nested for loop with 3 total loops instead of just 2.
- As we can see, the complexity of the array increases exponentially with the increase in dimensions.

Passing Multidimensional Array to a Function

We can also pass Multidimensional arrays as an argument to the function.

Example: Passing two-dimensional array to a function

```cpp
#include <iostream>
using namespace std;
// pass a 2-dimension array as a parameter
void display(int n[][2]) {
    cout << "Displaying Values: " << endl;
    for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 2; ++j) {
            cout << "num[" << i << "][" << j << "]: " << n[i][j] << endl;
        }
    }
}
int main() {

    // initialize 2-dimension array
    int num[3][2] = {
        {3, 4},
        {9, 5},
        {7, 1}
    };
    display(num);// pass a 2d array as an argument
    return 0;
}
```

```
Displaying Values:
num[0][0]: 3
num[0][1]: 4
num[1][0]: 9
num[1][1]: 5
num[2][0]: 7
num[2][1]: 1
```

- In the above program, we have defined a function named display(). The function takes a two dimensional array, int n[][2] as its argument and prints the elements of the array.
- While calling the function, we only pass the name of the two-dimensional array as the function argument display(num).

Note: It is not mandatory to specify the number of rows in the array. However, the number of columns should always be specified. This is why we have used int n[][2].

We can also pass arrays with more than 2-dimensions as a function argument.

Advantages of C++ Array

- Code optimization (less code)
- Random access
- Easy to traverse data
- Easy to manipulate data
- Easy to sort data etc.

Disadvantages of C++ Array

- Predetermining the size of the array is a must
- Fixed size (static structure)
- Modifying array elements are difficult and time consuming
- Chance of memory wastage or shortage

Exercises:

1. Find the largest element of a given array using passing array to function.
2. Find the smallest element of a given array using passing array to function.
3. Find the summation of two given arrays using passing array to function.

....